TCP Limit: A Streaming Friendly Transport Protocol

Felicián Németh, Sándor Molnár, Péter Tarján and Róbert Szabó
Department of Telecommunications and Media Informatics,
Budapest University of Technology and Economics,
H-1117, Budapest, Magyar Tudósok krt. 2
Fax: +36-1-463-1763
Email: nemethf@tmit.bme.hu

***Abstract*—In this paper we propose a simple enhancement to TCP (called TCP Limit) which significantly improves the utilization, delay, and jitter characteristics of standard TCP versions. TCP Limit requires only a small sender-side modification and can be applied on top of any TCP versions. Its main advantage is that it can be multiplexed with non-elastic streaming traffic without any router or protocol support. TCP Limit avoids the undesired effect of oscillation and keeps the delay and jitter as low as possible; so it does not ruin the performance of streaming applications. In the paper we present the initial performance evaluation of the proposed TCP Limit carried out by simulation and measurement. Analytical calculations are also included.**

***Index Terms*—Computer networks, Protocols, Congestion control, TCP**

I. Introduction

The Transmission Control Protocol (TCP) is the well known reliable data transfer protocol [1]; it is one of the key components of the current Internet architecture. However, the networking environment has gone through significant changes since the initial proposal of the first TCP version. The research community has already proposed modifications to the TCP congestion-control algorithm to achieve higher throughput in networks with large bandwidth-delay product, and/or wireless links or to adapt quickly to load fluctuations (e.g., HighSpeed TCP [2], Bic-TCP [3], TCP Westwood [4], [5], TCP Eifel [6], just to mention a few). Instead of incremental modifications, some authors redesigned the transport protocol from scratch (e.g., RCP [7], UDT [8], [9]).

Despite the numerous new algorithms, there is no consensus on the new ultimate transport protocol that would replace the traditional TCP. The lack of consensus is partially a result of lack of common performance evaluation metrics and evaluation models. To accelerate the process Sally Floyd collected and organized the possible evaluation metrics and the main open research issues in [10]. One of the aims of the current research clearly emphasized in [10] is to achieve stability in terms of minimizing oscillations of queuing delay or throughput.

The main problem is that rate fluctuations can result in router queue-size fluctuations, which may in turn cause frequent queue overflows as a key reason beyond loss synchronization. This phenomenon indicates major instability of the network. Our goal was to develop a stable TCP variant called TCP Limit which minimizes the oscillations in queueing delay and throughput.

TCP Limit achieves low delay jitter while remaining a sender-side only modification. The proposed TCP Limit works with small rate oscillations therefore can achieve high stability. Moreover, the mechanism reduces packet losses, which are widely used as an indicator of congestion; as a result we have a loss-based TCP with delay-based advantages.

The small delay and jitter of TCP Limit has a great advantage in the Internet where versatile streams are multiplexed: *it is friendly to streaming traffic*. The major problem identified in the current Internet is that elastic and non-elastic traffic cannot be efficiently managed without AQM (Active Queue Management) or other mechanisms, but these require router and/or protocol support. Since TCP Limit produce small delay and jitter, it cannot ruin the characteristics of non-elastic streams and can be multiplexed with streaming traffic without any router and/or protocol support. Subsequently we will clearly demonstrate this streaming-friendly property of TCP Limit.

TCP Limit can also work with 100% link utilization even in case of small buffers. As opposed to earlier TCP versions, TCP Limit can achieve this highly desirable operating point. It avoids buffer overflow hence multiple losses for steady states and also maintains full utilization of the bottleneck link. In addition, the limited congestion-window algorithm as part of the TCP Limit prevents the periodic oscillation of the sender's windows size. Therefore, flows experience stable round-trip time hence exhibit reduced jitter.

Another advantage of TCP Limit is that it requires minor sender-side modifications and can be applied to any TCP variants (e.g., NewReno TCP or any NewReno descendant like TCP Westwood). As a result, the advantages of any other TCP variant can be exploited in the TCP Limit. As an example, we show how TCP Limit can be applied to TCP Westwood.

The rest of the paper is organised as follows. Sec. II presents an overview of the related works pointing out an important design principle that our proposal is based upon as well. In Sec. III, we describe TCP Limit. We evaluate the

The work is supported by the 2/032/2004 ELTE-BME-Ericsson NKFP project on Research and Developments of Tools Supporting Optimal Usage of Heterogeneous Communication Networks

proposed method via laboratory measurements and simulations in Sec. V. Finally, Sec. VI draws the conclusion.

II. Prior Art

The congestion control algorithm of TCP NewReno consists of two main phases: *slow-start* and *congestion-avoidance*. In slow-start phase, the sender increases the congestion window (`cwnd`) exponentially until it reaches the slow-start threshold (`sstresh`). Afterwards, TCP switches to the congestion-avoidance phase where `cwnd` grows linearly until packet loss is detected. If the loss is discovered by duplicated acknowledgments, the lost packet gets retransmitted and the `cwnd` is halved (fast-recovery) and the flow returns to the congestion-avoidance phase. One of TCP's problems is related to the linear increase in the congestion-avoidance phase, which prevents fast utilization of the link bandwidth for high bandwidth-delay product (BDP) scenarios. Another problem relates to the setting of the initial `sstresh` value.

Dynamic bandwidth presents another challenge to TCP performance: if a large amount of bandwidth becomes available for reasons such as wide bandwidth-consuming flows leaving the network, then TCP may be slow in catching up. The congestion-avoidance phase is in additive-increase mechanism: `cwnd` is incremented only by one packet per round-trip time (RTT). As a result, lack of link utilization can be experienced.

The above problems have been addressed by successive refinements of TCP Westwood. Westwood [4], [5], [11] has the next important underlying design principle: it differs from the normal TCP behaviour only when other TCP connections do not notice the difference. TCP NewReno achieves low throughput in networks with high bandwidth-delay product mainly because its congestion-avoidance algorithm increases too conservatively. Instead of merely changing the congestion-avoidance algorithm into a more aggressive one like High-Speed TCP does, Westwood runs a Persistent Non-Congestion Detection (PNCD) method, which detects continuously available free bandwidth. Persistent non-congested situation is interpreted as being harmless to invoke Agile Probing. Agile Probing increases the sending rate more aggressively because no one else takes advantages of the available bandwidth. Both Agile Probing and PNCD are based on bandwidth estimations.

There is another extension in the Westwood proposals originally called Faster Recovery [12]. After having detected a packet loss, Faster Recovery uses bandwidth estimation instead of dumbly halving its `cwnd` to reset its `cwnd` and `sstresh`. They are reset to the *congestion window equivalent* (CWE) of the estimated bandwidth (BWE). CWE is calculated as:

$$\texttt{CWE}(\texttt{BWE}) = \frac{\texttt{BWE} * \texttt{RTT}_{\texttt{min}}}{\texttt{seg\_size}}, \tag{1}$$

or if `cwnd` is measured in bytes instead of packets then

$$\texttt{CWE}(\texttt{BWE}) = \texttt{BWE} * \texttt{RTT}_{\texttt{min}}, \tag{2}$$

where $\texttt{RTT}_{\texttt{min}}$ denotes the minimal measured RTT and `seg_size` is the segment size.

When the packet loss is a result of buffer overflow[1], equation (1) leads to `cwnd` halving, i.e., there is no noticeable difference from the standard TCP behaviour. On the other hand, if the packet loss is due to channel error, then traditional TCP treats this, once again, as a sign of congestion and halves its `cwnd`. Now halving `cwnd` is considered too drastic since the queue is not full. In this case, CWE results in a more moderate backoff.

Compound TCP [13] improves NewReno performance in high speed networks in a TCP-friendly manner. It extends the loss-based congestion control with a delay-based component. The delay-based part is responsible for the fast speed-increase and is only in use when it detects the path to be under utilized. This is the same principle that Westwood modifications and TCP Limit are based on. However Compound TCP aims to increase throughput in case of free bandwidth and not to minimize rate oscillation.

Leith et al. proposes a delay-based TCP variant in [14] with the main goal of achieving small buffer occupancy while the congestion window is updated in an additive-increase multiplicative-decrease manner. The backoff strategy of the protocol is essentially based on ideas similar to those of Faster Recovery and therefore shows similarities to the backoff of TCP Limit. As opposed to our proposal, it sacrifices full link utilization for intra-protocol friendliness.

III. Limited Congestion Window Algorithm

The main contribution of this paper is an algorithm that dynamically controls the size of the `cwnd` and limits its fluctuation by using time-tested bandwidth estimators. TCP Limit is a relatively small, sender-side-only modification of TCP NewReno, but can be applied, for example, on the top of TCP Westwood as well. Since its behaviour is indistinguishable from that of NewReno when the modification is inactive, it follows the design principle outlined previously. The modification is activated when the estimation of the achieved rate does not grow simultaneously with `cwnd`. In this case the algorithm inhibits `cwnd` from growing further. Although any other achieved rate estimation technique would do, we use the so-called Rate Estimation (RE) method of Westwood [11] as a demonstration. TCP Limit only modifies the congestion-control algorithm, when an acknowledgment is received in the congestion avoidance phase. First, we present a simplified version of the algorithm. The modified rather simple pseudo code of the method is given by

```
1 if cwnd > CWE(RE) * α then
2    do nothing
3 else
4    cwnd ← cwnd + NRI(cwnd)
5 end if.
```

RE: achieved rate estimation
CWE: cong.window equivalent, eq. (1)
NRI: NewReno increment

If the congestion window is much larger than the congestion window equivalent of the estimated achieved rate (line 1, α should be larger than 1), then the `cwnd` is kept constant (line 2), otherwise we increase the `cwnd` just like the normal

[1] and the bottleneck queue is sized to the bandwidth-delay product.

NewReno would do (line 4). NRI(x) and x denote the increase used by NewReno when it is in the congestion-avoidance phase and the size of its congestion window, respectively[2].

Basically, cwnd is not allowed to substantially exceed the achieved rate. The basic idea of TCP Limit is the following: (i) if the cwnd is larger than the CWE of the achieved rate, then increasing the cwnd only fills up the bottleneck queue; meanwhile (ii) keeping the cwnd constant avoids the oscillation of the flow.

The reason of using the constant α is twofold. First, the rate estimation might not be always accurate. Second, by limiting cwnd above CWE, the flow can instantly increase its achieved rate when free bandwidth appears in the bottleneck link as demonstrated in Fig. 1. Parameter α is set to 1.1, the maximal transfer unit is 1500 bytes, and bottleneck queue is sized to BDP throughout the paper. The figure shows the start of a TCP Limit flow. It enters into congestion-avoidance phase slightly before $1s$. After that, its cwnd linearly increases until about $4s$ when the achieved rate stops growing because the path has become saturated. Then the Limit algorithm prohibits the cwnd from growing further. At $8s$ the background traffic leaves the system and the RE of the TCP flow starts to increase implying cwnd growth. 1.5 seconds later the RE stops increasing, now because the TCP flow alone fully utilizes the link; the cwnd continues growing until about $9.5s$ when cwnd reaches α times the achieved rate. So the transmission speed becomes constant for the second time.

As shown in Fig. 1, the simple algorithm outlined above avoids the well known "saw-tooth" oscillation pattern of NewReno. However, it decreases its cwnd only when it detects a packet loss. As a consequence, this method leaves no room for a newcomer TCP Limit flow. In order to achieve fair bandwidth utilization between TCP Limit flows, we extended the simple algorithm by forcing it to decrease the cwnd recurrently. One key component of approaching fair utilization is the frequency of the forced decreases. The pseudo code of the extended algorithm is as follows:

```
1  v_cwnd ← v_cwnd + NRI(cwnd)
2  v_target ← min(v_target, 2 ∗ CWE(NL) − CWE(RE))
3  if cwnd > CWE(RE) ∗ α then
4     if v_cwnd > v_target then
5        cwnd ← CWE(RE)
6        v_cwnd ← CWE(RE)
7        v_target ← CWE(RE)
8     end if
9  else
10    cwnd ← cwnd + NRI(cwnd)
11 end if.
```

v_cwnd: virtual congestion window
v_target: virtual target
NL: narrow link capacity

We always maintain a virtual congestion window variable (v_cwnd, line 1), which initially equals to cwnd, but it is increased even if cwnd is kept constant. When v_cwnd reaches a certain level, v_target (line 4), then cwnd (and v_cwnd) are decreased to the achieved rate in order to give opportunity to the competing flows to grow (line 5–7). The

[2]NRI(x)=seg_size∗seg_size/x when the congestion window is measured in bytes [15]

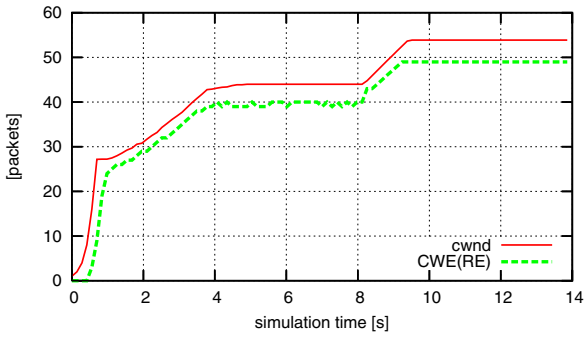


Fig. 1. Simulation result of a simple TCP Limit flow traversing a link of 5Mbps bandwidth and 60ms latency. The TCP flow competes a constant bit rate traffic (CBR) of 1Mbps until 8s. Initial slow start threshold is 25 packets.

NL in line 2 denotes the estimate of the narrow link[3] capacity. Line 2 and line 4 help the throughput of the TCP Limit flows converge: the faster a flow transmits, the smaller v_target is. This latter results in a more frequent decrease of its cwnd. (On the other hand, in line 5, cwnd is reduced to RE, therefore the achieved rate of the flow will not be decreased when there is no other flow competing with it.) After the forced decrease, equation in line 3 does not hold anymore; so the flow starts increasing its cwnd linearly.

In our simulations and measurements a maximum filter on Bandwidth Estimation (BE) [4] has been used to estimate the narrow link. BE is known to be inaccurate under some circumstances [16]. As a solution, BE can be replaced with any other narrow-link-estimation algorithm such as the much more robust TCPProbe [17]. Note that, however, the accuracy of the narrow link estimation has only indirect influence on the occurrence of forced cwnd decreases.

Fig. 2 illustrates the behavior of the forced decrease. At the beginning, the TCP Limit flow alone fully utilizes the link, hence RE equals to NL, so cwnd is reduced right after the equation in line 2 becomes true. At $15s$ a bandwidth-consuming flow enters the system and therefore RE falls. The smaller RE is, the larger v_target grows, and therefore the less frequent the forced decreases occur. After each of the two backoffs in $(20s, 35s)$ the cwnd regains its previous level, since the CBR flow does not take advantage of the free resources. Note that v_cwnd reaches v_target at approximately $37s$, however forced decrease happens only when the equation in line 3 first holds shortly before $40s$. v_target can only decrease between backoffs (line 2): if RE increases, then backoffs should be more frequent; on the other hand, cwnd and queue occupancy do not decrease with decreasing RE.

IV. Queue Length Analysis

We assume that packets are sent over a single bottleneck link with a speed of μ packets per second. Let T denote the base RTT, then the pipe capacity is $C = \mu T$ and the maximal buffer size $Q = W^{\max} - C$, where the maximal cwnd size

[3]Narrow link of a path is the link with the smallest capacity.

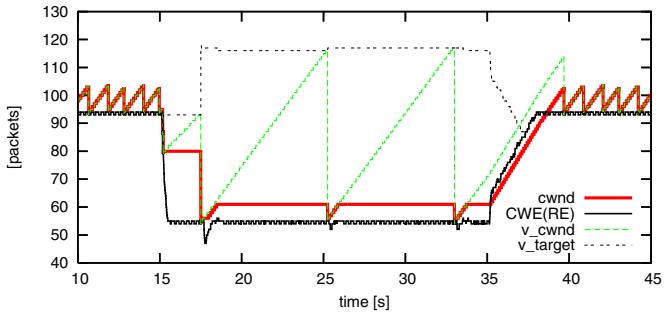


Fig. 2. Measurement result of a TCP Limit flow traversing a bottleneck link of 10Mbps bandwidth and 50ms latency. 4Mbps CBR background traffic appears in $[15s, 35s]$. Note that, `v_cwnd` and `cwnd` coincide and `v_target` and CWE(RE) also coincide during several intervals.)

that can be achieved by the algorithm is $W^{\max} = \alpha$ CWE(RE). Using (2) gives CWE(RE) = CWE(μ) = μT. From the last three equations the maximal buffer size will be $Q = C(\alpha - 1)$.

Similarly, if we have a set of connections on a bottleneck link, then the buffer size clearly becomes

$$Q = \sum_{i=1}^{N} q_i, \tag{3}$$

where q_i is the backlog belonging to flow i and N is the number of competing flows. However, q_i can be expressed in TCP Limit as

$$q_i = W_i^{\max} - c_i, \tag{4}$$

where c_i is the pipe's capacity share of flow i. In the simplest case when $\alpha_i = \alpha$ and the RTTs are the same for each flow ($T_i = T$), then

$$W_i^{\max} = \alpha_i \text{ CWE(RE}_i) = \alpha_i \mu_i T_i, \tag{5}$$

which yields

$$Q = \sum_{i=1}^{N} (W_i^{\max} - c_i) = \alpha T \sum_{i=1}^{N} \mu_i - \sum_{i=1}^{N} c_i. \tag{6}$$

In case of a single bottleneck topology $\sum c_i$ is at most C. Moreover, the sum of flow transmission rates is the bottleneck bandwidth ($\sum \mu_i = \mu$) in the worst-case; hence the overall maximal backlog in the bottleneck link becomes

$$Q = C(\alpha - 1). \tag{7}$$

As seen, the backlog in the bottleneck link will be constrained as if a single flow was in the system independently of the number of competing connections. As a result, the backlog will be only a portion of the bandwidth-delay product. More precisely (7) shows that the queue cannot grow beyond $(\alpha - 1)$ times the bandwidth-delay product.

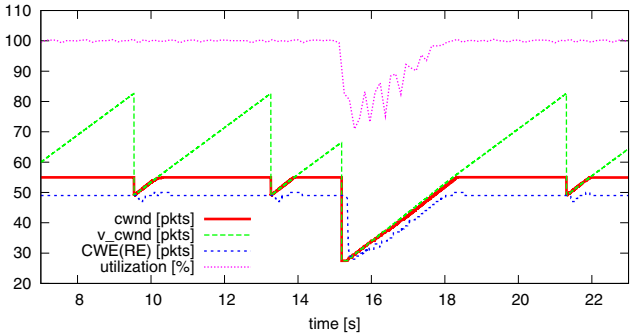
V. Numerical Results

All the simulation results in this paper were obtained using ns-2 [18]. The measurements were carried out in a testbed. The PCs in the testbed ran Linux 2.6.16 and had CPUs of 3 GHz Pentium 4, 2 GB RAM and Intel e1000 PCI-Express network cards. The router at the bottleneck link emulated different link delays, bandwidth, and queue sizes through the tc (traffic control) interface using the *netem* (network emulator) and *tbf* (token bucket filter) modules. TCP flows were started by *netcat* (nc). Since the available RE implementation in Westwood+ [19] calculates bandwidth samples only once per RTT, we implemented the RE bandwidth estimator and the TCP Limit.

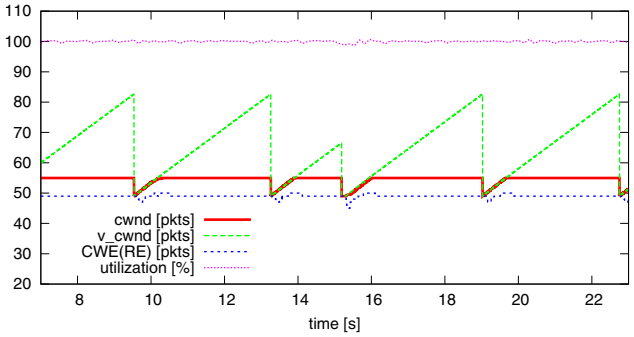
A. TCP Limit on the top of TCP Westwood

Since TCP Limit modifies NewReno in a well-defined and isolated part, it can applied to not only NewReno but other TCP versions as well. As an example, we show how it can be combined with one of the extensions of the Westwood proposals.

Faster Recovery does not halve the `cwnd`, hence it maintains full utilization even when (i) a loss happens due to congestion and the bottleneck queue size is smaller that BDP, or (ii) channel error causes packet loss. TCP Limit operates with full utilization even when queues are not sized to BDP. However, random loss degrades its performance (Fig. 3(a)). Fig. 3(b) illustrates the benefit of combining the Faster Recovery modification of Westwood with TCP Limit. The advantage can be clearly seen: the unified TCP can keep full utilization even if we expose it to artificial packet loss. Therefore, we recommend extending the TCP protocol stack with the TCP Limit algorithm if it contains Westwood.



(a) TCP Limit without Faster Recovery



(b) TCP Limit With Faster Recovery

Fig. 3. Simulation result: artificial packet drop at 15s.

B. Fairness

Fairness is measured by the relative performance of competing flows of the same TCP variant. Fig. 4 shows the result of an experiment with 2 connections competing. The slow-start threshold is set practically to infinity in the NewReno implementation. Therefore, when the second flow starts it causes and suffers from multiple losses. TCP Limit extends NewReno, hence both flows lose packets at about $16s$. (The losses at the beginning of the connection's lifetime are the result of the overly aggressive slow-start phase. These losses can be avoided by using the Agile Probing method of Westwood. Although, it has not yet been incorporated into the Linux code base.) As shown in Fig. 4, the backoff frequency of flow 2 remains larger than that of flow 1 until the two congestion windows converge to each other, and therefore the fair utilization is gradually approached.

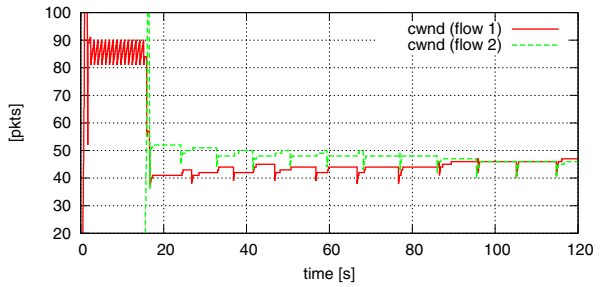


Fig. 4. Measured result: two TCP Limit flows compete in a link of 10 Mbps and 100ms.

To numerically evaluate the fairness, we carried out 20 experiments and calculated Jain's fairness indices[4] in the equilibrium state. Average fairness index of 20 experiment is 0.957 with standard deviation of 0.044. In a similar experiments with NewReno, the average fairness index was 0.994 with standard deviation of 0.005. We can conclude that there is a minor degradation of the faireness performance of TCP Limit compared to NewReno. This is the price to be payed for the smaller delay, smaller delay jitter, and for the other benefits.

C. Friendliness to streaming traffic

Due to TCP's oscillating nature, it can harm the quality of delay- or jitter-sensitive traffic when their paths have common links. A general solution is to treat the connections differently at routers, e.g., to give higher priority to streaming traffic. However, special treatment is not always feasible.

We conducted a test lab measurement where three connections are treated equally in their common bottleneck router (Fig. 5). One of the connections is a large CBR traffic representing many voice over IP (VoIP) traffic sources multiplexed. The other one is a real VoIP stream generated by WinRTP [21]. The third connection is an infinite download using traditional TCP NewReno at the first run. The experiment was repeated secondly with the TCP protocol exchanged with TCP Limit. The propagation delay of the bottleneck link was adjustable.

[4]Jain's fairness index f is defined in [20]. In case of two flows $f = 0.5(\mu_1 + \mu_2)^2/(\mu_1^2 + \mu_2^2)$, where μ_i is the throughput of flow i.

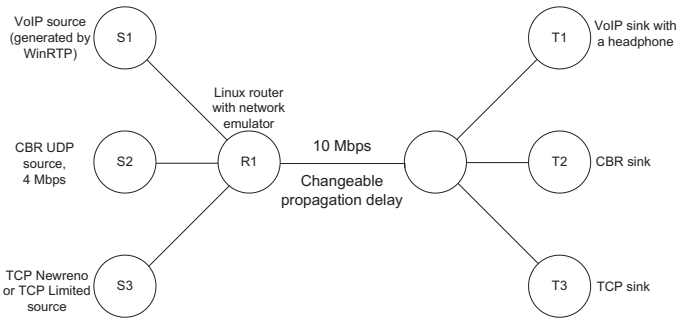
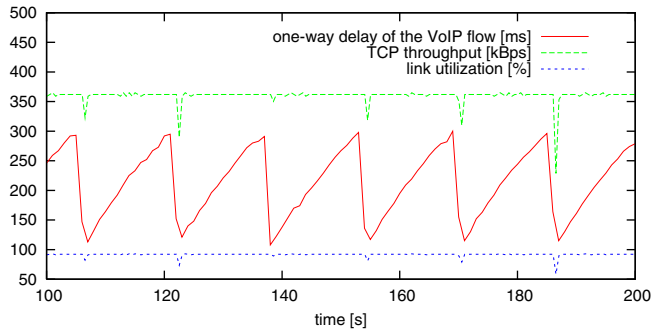
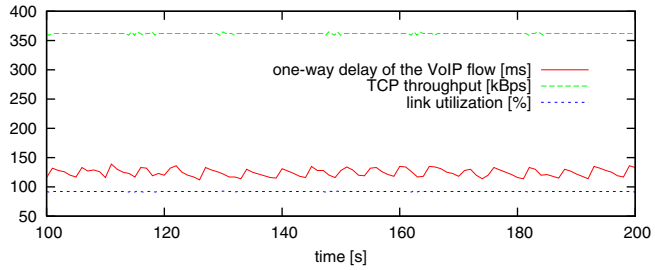


Fig. 5. Topology of the friendliness experiment.



(a) TCP NewReno



(b) TCP Limit

Fig. 6. Measured result: a TCP flow competes with a VoIP stream in a link of 10 Mbps and 100ms.

Fig. 6 shows the delays the VoIP stream undergoes in the two cases. The results clearly show that both the maximal delay and the delay jitter are significantly lower in case of our proposed TCP Limit resulting in better voice quality for the VoIP application. We could observe crackling sound right after the backoffs of NewReno. On the other hand, TCP Limit produce small delay and jitter and it cannot ruin the characteristics of the non-elastic stream.

The NewReno flow increases its `cwnd` until it detects a packet loss. Loss happens when the awaiting packets fully fill up the bottleneck queue. However, the router puts the VoIP packets and the TCP packets in the same queue, hence the VoIP queuing delay follows the delay of the TCP flow. Therefore, the VoIP delay resembles the saw-tooth pattern of NewReno achieving delay as much as 300ms (Fig. 6(a)) but it remains below 150ms in the case of TCP Limit (Fig. 6(b)).

We repeated the above experiment with different link delays

(Fig. 7). TCP Limit has remarkably less influence on the VoIP flow in terms of delay jitter, mean delay, and maximal delay. The importance of the figure comes from the fact that the tolerable delay in human conversation is about 200ms: the VoIP flow performs much worse in this regard when competing with NewReno.

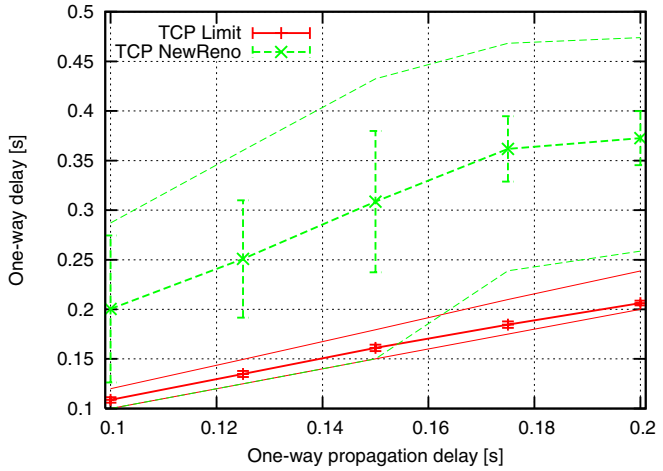


Fig. 7. Simulation result: one-way delays of a VoIP flow when competing a TCP flow: maximum, mean with relative variance, and minimal delays.

D. Performace comparison with other TCP variants

Recently the research community has started to identify performance metrics which properly characterize the performance of transport protocols [10]. In addition, automatic evaluation tools are being developed that calculate these metrics and profoundly compare the protocols. This section presents results obtained by the tool of Wang et al. [22].

The experiment is based on a dumb-bell topology. There are five FTP connections using the analyzed protocol both in the forward and in the reverse paths. The background traffic consists of streaming video, interactive voice, and short-lived web traffic using normal TCP. The reader is referred to [22] for details. We excluded those protocols that depend on extra functionality in network routers (VCP, XCP). We repeated each random simulation 15 times and averaged the results.

Fig. 8 shows link utilization, queue occupancy, and packet drop rate when the bottleneck capacity varies from 1 Mbps to 1000 Mbps. As shown, the link utilization of TCP Limit and the other variants[5] have similar characteristics. However, the mean queue length and therefore the average queuing delay are the smallest among the TCP versions when the bottleneck capacity is larger than 10 Mbps. The mean queue length of TCP Limit is considerably smaller at every link speed than that of the rest of the investigated protocols. Furthermore, it is the TCP Limit for which the loss rate diminishes to 0 at the lowest link speed.

[5]CUBIC, HighSpeed TCP, Hamilton TCP, TCP NewReno, TCP SACK, Scalable TCP, TCP Westwood

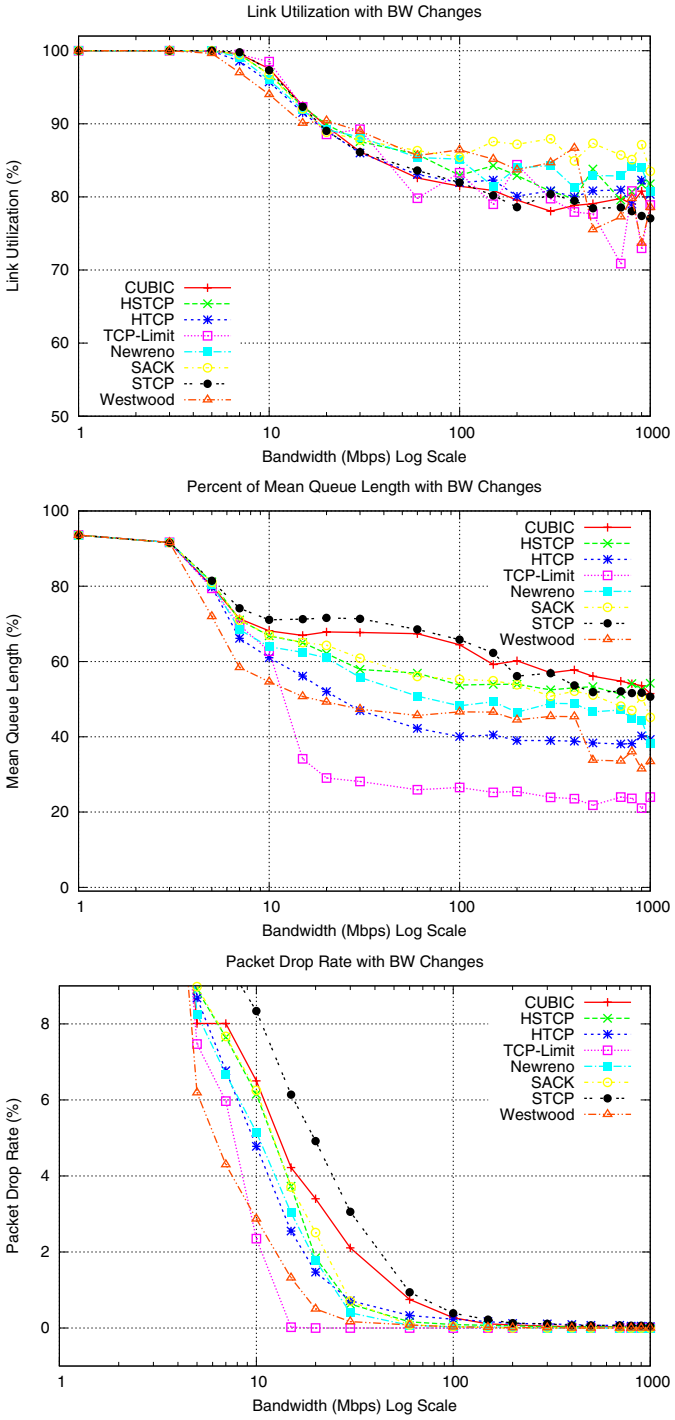


Fig. 8. Utilization, queue length and drop rate with changing bandwidth (obtained with the help of [22])

VI. Conclusion

In this paper we presented a novel enhancement to the TCP congestion control algorithm. Our modification (TCP Limit) dynamically limits the size of the congestion window when it is considerably larger than the achieved rate and keeps the bottleneck queue from increasing. On the other hand, TCP Limit keeps the congestion window constant to avoid the undesired effect of TCP oscillation.

To summarize the advantages, the proposed TCP-Limit algorithm

- works with full utilization even in case of small buffers,
- reduce packet loss,
- drastically alleviate the oscillation effect without AQM mechanisms,
- achieves low delay characteristics,
- achieves low jitter characteristics,
- has a streaming friendly property,
- requires minor sender-side modification of TCP versions,
- can be applied to any TCP variants.

The advantages of TCP Limit were confirmed in part theoretically and in part by simulation- and measurement-based performance evaluations. We illustrated how TCP Westwood can benefit from implementing TCP Limit on the top of it. The streaming friendly property of TCP Limit was demonstrated in a testbed environment and we have shown that a VoIP application can work together with TCP Limit due to its improved delay and jitter characteristics.

The performances of TCP Limit and other TCP variants have been compared. We have found that TCP Limit exhibits substantially improved performance in terms of queue length and packet drop rate characteristics.

Future work includes parameter fine-tuning, experimentation with different bandwidth estimators, evaluation of complex topologies and traffic mixes. In addition, the effect of active AQM mechanisms on TCP Limit should also be addressed.

References

[1] V. Jacobson, "Congestion avoidance and control," in *ACM SIGCOMM '88*, Stanford, CA, Aug. 1988, pp. 314–329.

[2] S. Floyd, "HighSpeed TCP for large congestion window," RFC 3649, Dec. 2003.

[3] L. Xu, K. Harfoush, and I. Rhee, "Binary increase congestion control for fast long-distance networks," in *Proc. of IEEE Infocom*, 2004.

[4] S. Mascolo, C. Casetti, M. Gerla, M. Y. Sanadidi, and R. Wang, "Tcp Westwood: Bandwidth estimation for enhanced transport over wireless links," in *Proc. of ACM MOBICOM'01*, Jul. 2001.

[5] R. Wang, K. Yamada, M. Y. Sanadidi, and M. Gerla, "Tcp with sender-side intelligence to handle dynamic, large, leaky pipes," *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 2, pp. 235–248, Feb. 2005.

[6] R. Ludwig, "The Eifel algorithm for TCP," Internet draft, draft-ietf-tsvwg-tcp-eifel-alg-00.txt, Feb. 2001. [Online]. Available: http://iceberg.cs.berkeley.edu/downloads/tcp-eifel/draft-ietf-tsvwg-tcp-eifel-alg-00.txt

[7] N. Dukkipati and N. McKeown, "Why flow-completion time is the right metric for congestion control," *ACM SIGCOMM Communication Review*, vol. 36, no. 1, pp. 59–62, Jan. 2006.

[8] Y. Gu and R. L. Grossman, "Optimizing UDP-based protocol implementation," in *Proc. of PFLDNet 2005*, Lyon, France, Feb. 2005.

[9] ——, "UDT: A transport protocol for data intensive applications," Internet draft, draft-gg-udt-01.txt, Aug. 2004. [Online]. Available: http://udt.sourceforge.net/doc/draft-gg-udt-01.txt

[10] S. Floyd, "Metrics for the evaluation of congestion control mechanisms," Internet draft, Mar. 2007, draft-irtf-tmrg-metrics-09. [Online]. Available: http://www.icir.org/tmrg/draft-irtf-tmrg-metrics-09.txt

[11] R. Wang, M. Valla, M. Sanadidi, B. Ng, and M. Gerla, "Efficiency/friendliness tradeoffs in TCP Westwood," in *Proceedings of IEEE/ISCC 2002*, Taormina, Italy, Jul. 2002.

[12] C. Casetti, M. Gerla, S. Lee, S. Mascolo, and M. Sanadidi, "TCP with faster recovery," in *Proceedings of the IEEE Military Communications Conference (MILCOM 2000)*, Los Angeles, CA, USA, Oct. 2000.

[13] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "Compound tcp: A scalable and tcp-friendly congestion control for high-speed networks," in *4th International Workshop on Protocols for Fast Long-Distance Networks (PFLDNet)*, Nara, Japan, 2006.

[14] D. Leith, R. Shorten, G. McCullagh, J. Heffner, L. Dunn, and F. Baker, "Delay-based AIMD congestion control," in *Proc. of PFLDnet*, Feb. 2007.

[15] W. Stevens, "TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms," Internet Engineering Task Force, RFC 2001, Jan. 1997.

[16] A. Glowacz and R. Chodorek, "Behavior of tcp westwood in wireless network," in *Proc. Advanced Technologies, Applications and Market Strategies for 3G and Beyond ATAMS'2002 International Conference*, Cracow, Poland, Dec. 2002.

[17] C. Marcondes, A. Persson, L.-J. Chen, M. Y. Sanadidi, and M. Gerla, "TCP Probe: A TCP with built-in path capacity estimation," in *The 8th IEEE Global Internet Symposium (in conjunction with IEEE Infocom.05)*, Miami, USA, 2005.

[18] "The network simulator ns-2," http://www.isi.edu/nsnam/ns/.

[19] R. Ferorelli, L. Grieco, S. Mascolo, G. Piscitelli, and P. Camarda, "Live internet measurements using Westwood+ TCP congestion control," in *Proc. of Globecom'02*, vol. 3, Nov. 2002, pp. 2583–2587.

[20] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling*. New York: Wiley, 1991.

[21] Vovida Networks, "WinRTP: Audio RTP library for windows," available from http://www.vovida.org/applications/downloads/winRTP.

[22] G. Wang, Y. Xia, and D. Harrison, "An NS2 TCP evaluation tool," Internet Engineering Task Force, Internet Draft, draft-irtf-tmrg-ns2-tcp-tool-00.txt, Apr. 2007.